CHAPTER 5

POWER MODELING

Jason Mair[1], Zhiyi Huang[1], David Eyers[1], Leandro Cupertino[2], Georges Da Costa[2], Jean-Marc Pierson[2] and Helmut Hlavacs[3]

[1]Department of Computer Science, University of Otago, New Zealand
[2]Institute for Research in Informatics of Toulouse (IRIT), University of Toulouse III, France
[3]Faculty of Computer Science, University of Vienna, Austria

5.1 Introduction

Power consumption has long been a concern for portable consumer electronics, with many manufacturers explicitly seeking to maximize battery life in order to improve the usability of devices such as laptops and smart phones. However, it has recently become a concern in the domain of much larger, more power hungry systems such as servers, clusters and data centers. This new drive to improve energy efficiency is in part due to the increasing deployment of large-scale systems in more businesses and industries, which have two primary motives for saving energy. Firstly, there is the traditional economic incentive for a business to reduce their operating costs, where the cost of powering and cooling a large data center can be on the order of millions of dollars [18]. Reducing the total cost of ownership for servers could help to stimulate further deployments. As servers become more affordable, deployments will increase in businesses where concerns over lifetime costs previously prevented adoption. The second motivating factor is the increasing awareness of the environmental impact—e.g. greenhouse gas emissions—caused by power production. Reducing energy consumption can help a business indirectly reduce their environmental impact, making them more clean and green.

The most commonly adopted solution for reducing power consumption is a hardware-based approach, where old, inefficient hardware is replaced with newer, more energy effi-

cient alternatives. Hardware upgrades provide a means of improvement which is transparent to end users, requiring no changes in user behavior. Despite this convenience, there are several key drawbacks to such hardware deployment. The foremost problem is the significant financial cost associated with upgrading hardware, which will be substantial for any large deployment. Moreover, such upgrades challenge sustainability, creating significant amounts of material waste that is hard to handle. Furthermore, efficiency improvements are often mistakenly equated with a corresponding reduction in total power consumption [24]. This is commonly not the case, as efficiency improvements have a tendency to increase the scale of system deployments, resulting in greater energy consumption [33]. Previously, such impacts were exacerbated by datacenter over-provisioning, which resulted in a substantial amount of idle resources [2].

As an alternative, software usage can be explored to reduce energy consumption. Due to power savings mechanisms present on newer hardware, software execution plays a big role on the total dissipated power in computers. Since there is no technique to directly measure the power of an application, this approach requires an applications power estimator development through the use of power measurements in a feedback mechanism. Such estimator needs to provide power values at a significantly finer granularity than hardware power metering allows, i.e. providing application or component specific power consumption statistics. By incorporating power measurements, software policies will be capable of quantitatively evaluating the runtime tradeoffs made between power and performance for a given configuration selection.

In addition to improving the effectiveness of many existing power saving policies, such as workload consolidation within a datacenter, power models allow new policies to be developed that were not possible previously. One such policy is the implementation of software-based power caps, where tasks are free to execute on a system as long as total power consumption remains below a set threshold. When exceeded, applications will be progressively throttled, or stalled until power consumption decreases. Such a policy can be applied at a finer granularity, enforcing caps specific to applications, potentially reining in rogue applications that are prone to excessive power consumption. A further policy, proposed by Narayan and Rao [28], is to charge datacenter users for power use, similarly to how other resources would be charged. These are but two of the possible use-cases, with more discussed later in the chapter, none of which could be realized without the inclusion of accurate, fine-grained power estimation models.

Advanced power saving policies commonly require application specific power values for determining optimal configurations at runtime. Power meters have traditionally been used to provide application specific power measurements in single processor systems, as only one task was able to execute at any given time. However, modern multi-core processors present a more significant challenge, where system resources can be shared between a large number of concurrently executing tasks. As will be discussed in the next section, hardware power metering has not kept pace with the new processor developments as these meters are not capable of providing the fine-grained power measurements required to isolate individual applications. Therefore, techniques for power estimation models have been proposed that seek to quantify the relationship between application performance, hardware utilization and the resulting power consumption.

It has long been understood that a close, fine-grained relationship exists between modest changes in application performance and power use. This is intuitive as the power use of a system is strictly dependent upon the utilization levels of each component. It is this very relationship which is often leveraged in power saving policies, designed to lower utilization levels of key components in response to changes in execution workloads. The most

prominent policies to enact these responses are processor governors, which utilize hardware based Dynamic Voltage and Frequency Scaling (DVFS). Power estimation models attempt to quantify such changes i.e. determine the change in power consumption for a corresponding variation in component utilization (performance).

For example, if the running software is CPU intensive, the processor will have a high utilization level and a correspondingly high power level. Alternatively, if a large number of cache-misses are observed, memory accesses will be high, causing processor stalls and low utilization, resulting in a lower overall power draw. In quantifying the strength of the relationships between key performance events and power, an analytical model can be derived, enabling accurate runtime power estimation.

In this chapter we present how application or system's power models are created. First, section 5.2 describes different types of power metering techniques. Next, section 5.3 describes the most used performance indicators and how they are implemented. Then, section 5.4 relates performance and power, showing how performance impacts the power dissipated on some devices based on their polices and specifications. The main aspects of power modelling are briefly described in section 5.5, followed by a detailed description of each of such aspect, such as variable selection, learning algorithms, model evaluation and errors. Later, some use cases and experimental results are presented in section 5.6 along with a list of available softwares (section 5.7). Finally, some conclusions are drawn in section 5.8.

5.2 Measuring Power

Power meters can be divided according to its location, into two major classes: external or internal to the compute box. External meters are put in between computer's power plug and the socket-outlet, while internal ones are inside the computing box and can be device specific. Despite hardware power meters providing the most accurate source for system power measurements, they are incapable of providing the fine-grained, application specific values required by some power-aware policies. Consequently, power measurements are used as the target while creating a power estimation model, acting as an explanatory variable.

5.2.1 External Power Meters

The most commonly used method for monitoring runtime system power is through the use of an external, hardware power meter. For small scale deployments, i.e. a standalone system, commodity wall connected power meter, like the Watts Up? Pro[1] and iSocket[2] (InSnergy Socket) are used. Alternatively, large-scale, datacenter deployments can use intelligent power distribution units (PDU), which are standard rack PDUs with the additional capability of independently monitoring power for each connected machine. Despite incurring an additional purchase/upgrade cost for hardware, power meters have the significant advantage of easy deployment, requiring no alterations to be made to existing equipment or infrastructure.

However, power meters have two drawbacks regarding the granularity of results. First, the sample rate of power, typically once a second, is insufficient for detailed, fine-grained power analysis of application characteristics [14]. Second, power meters only return coarse-

[1] http://www.wattsupmeters.com
[2] http://web.iii.org.tw

grained power values, for the entire system, making it impractical to determine relative power use of individual applications or system components. Without this capability, power management policies can only be acted upon the entire system, rather than individual applications. This becomes a more significant restriction as processor core counts increase, allowing for more concurrently executing applications. Therefore, another solution is required for fine-grained power saving policies.

5.2.2 Internal Power Meters

A solution to the problem of coarse-grained power measurements is to embed lower level power sensors inside a system, enabling the isolation of component specific power use. Fine-grained measurements are possible by independently monitoring the DC power rails supplying power to each system component at the required voltage. Two of the most commonly used metering techniques are shunt resistors and clamp meters. Shunt resistors are placed inline for each power rail and measure the voltage drop across the resistor, allowing the current and power to be calculated. For easier deployment, clamp meters can be placed around each power rail, using Hall effect sensors to measure power. Despite such techniques being used during product development and testing of system components, manufacturers rarely incorporate internal meters into commodity products, partly due to concerns of additional cost [26].

However, since the Sandy Bridge microarchitecture, Intel has begun to embed power sensors in new processor designs, making predicted power values available through hardware registers. Currently, power measurements are restricted to a selection of three granularities: the entire processor package, power for all cores, or integrated graphics power [30]. System wide measurements can be made using third party solutions, like PowerMon2[3], which uses inline metering to monitor all system components using the 3.3V, 5V and 12V power rails. More specific monitoring is available through the NI[4] and DCM[5] meters, which only meter the 12V rail powering the processor. This highlights one of the key limitations of internal metering, that is, not all system components are able to be monitored. Components such as the power supply unit are outside the scope of metering, while not all solutions provide enough sensors to build a full system power model.

Despite the significantly finer granularity of power measurements, internal meters are incapable of isolating application specific power for shared resources. For example, multicore processors are powered by a single power rail, meaning individual core consumption is indistinguishable. Furthermore, placing additional equipment within a system can disrupt the airflow, causing increases in system temperature and a corresponding increase in power use [27]. This is not to mention the added inconvenience and potential cost of a manual deployment of metering equipment inside existing system deployments.

5.3 Performance Indicators

Performance values are the second key input parameter for power modeling, which are used as a proxy for hardwares utilization levels. During training, performance events provide the independent variables to be correlated with power observations. For the derived

[3]http://github.com/beppodb/PowerMon
[4]http://www.ni.com
[5]A non-commercial power meter by Universitat Jaume I

model, performance events are the only source of input used for power estimation, requiring sample rates to match runtime power values. Therefore, performance event selection is not only dependent upon correlation strength, but additionally on potential restrictions imposed by the expected execution environment. Fortunately, many alternative methods exist for monitoring application performance, where each has been designed to serve a different purpose. Techniques range from alternating applications and the corresponding execution, to monitoring low-level hardware events to unobtrusively glean performance insights at runtime. A range of such monitoring methods are presented in this section, enabling power model deployment across varied environments.

5.3.1 Source Instrumentation

The most commonly used technique for performance analysis is source code instrumentation, where additional instructions are inserted into the source code at key points of interest in order to provide detailed information during execution. Typical performance information can include data values or the execution time of code segments or functions. This performance data is mainly used during application development and testing in order to determine any points of execution bottleneck, isolating where a developer's efforts should be focused to improve an application's overall performance.

Source code instrumentation can either be achieved by manual code instrumentation, where instructions are selectively placed by the developer in a few key points of interest, or more extensively through the use of compiler tools. Gprof [17] is one such tool, capable of generating function call graphs, and determining the execution time for each function and its corresponding children.

Unfortunately, this dependence on the availability of application source code imposes some significant limitations on the usability of the approach. First, the instrumentation of code can alter the execution characteristics of an application [19], adding overhead that increases execution effort, and thus cause more energy to be used. Second, the performance statistics themselves are not sufficient to isolate component utilization levels. For this technique to be usable in power estimation it would need to be supplemented with additional sources of performance data, such as hardware performance counters, which will be discussed shortly.

5.3.2 Binary Instrumentation

Binary instrumentation allows for analysis functions to be inserted into an application binary, providing instrumentation of applications whose source code is not available. The Pin tool from Intel [25] is able to achieve this by using JIT (Just in Time) recompilation of the executable, whereby a user's predefined analysis routine is executed within the source binary. Instrumentation routines can be triggered when predetermined conditions are met. For instance, these can occur when: a specific function is called, a new function is called or on memory writes. Pin has been written such that it can be used with architecture independent profiling tools.

However, binary instrumentation suffers from similar limitations to source code instrumentation, with the primary concern being runtime overhead. For binary instrumentation, this can be even more significant, as a persistent overhead is incurred due to recompilation, which has been measured to be 30% before the execution of any analysis routes. While it has been proposed that such overheads can be reduced by limiting the profiling time by

dynamically attaching and detaching Pin, this is not sufficient for our use case of requiring runtime performance analysis to allow for persistent power estimation.

5.3.3 Performance Monitoring Counters

Performance Monitoring Counters (PMCs) are a set of special purpose hardware registers designed to monitor and record low-level performance events occurring at runtime in the processor. Such low-level performance events can include cache misses, processor stalls, instruction counts and retired operations. This allows for detailed insights into the utilization of different regions within a processor's hardware that are not possible with other monitoring techniques. Furthermore, the hardware-specific nature requires PMC registers to be placed on a per-core basis, enabling fine-grained performance monitoring within shared resources i.e. multi-core processors.

However, the use of hardware-specific performance monitoring can give rise to some additional challenges, where different manufacturers, or even micro-architecture versions, will support a different number of events and hardware registers, e.g. both AMD and Intel have their own model-specific PMC specifications [1, 22]. The AMD 10h microarchitecture supports four performance registers whose value can be selected from the 120 available performance events, where the Intel Sandy Bridge microarchitecture supports eight general-purpose performance counters per core, or four per-thread if two-way multithreading is used.

Tools such as *perf*[6] and *PAPI*[7] have been developed to help with some of these hardware dependence issues by providing a standard interface to the user regardless of the underlying microarchitecture. While microarchitecture support may vary, many of the main performance events are supported across architectures, regardless of how they are implemented. Table 5.1 lists some of *perf*'s available PMCs as described in its manual.

5.3.4 Operating System Events

The operating system is in a unique position to oversee all of the component interactions within a system, placing it in the best position to provide a broad picture of system performance. While the PMCs are able to provide fine-grained, low level performance events for the processor during execution, these details do not allow for insights into the performance of other components, such as the hard drive or network card. Much of the interaction between components is handled by the OS kernel, which is outside the scope of user-space monitoring tools. However, some of the OS events require kernel patches/modules to be accecible from the user perspective.

Therefore, tools have been developed, such as *Oprofile*[8] and *netatop*[9], which are capable of monitoring operating system kernel events during runtime. These tools enable the monitoring of events, such as system calls, interrupts, bus and network utilization. Such features are crucial for modeling system components other than the processor and memory, which have traditionally not been the focus of detailed performance analysis.

[6] https://perf.wiki.kernel.org/index.php/Main_Page
[7] http://icl.cs.utk.edu/papi/
[8] http://oprofile.sourceforge.net
[9] http://www.atoptool.nl/downloadnetatop.php

Table 5.1 Some of the hardware performance counters available in perf.

Name	Description
cpu-cycles	Total cycles (affected by CPU frequency scaling).
instructions	Retired instructions.
branch-instructions	Retired branch instructions.
branch-misses	Mispredicted branch instructions.
bus-cycles	Bus cycles, which can be different from total cycles.
idle-cycles-frontend	Stalled cycles during issue.
idle-cycles-backend	Stalled cycles during retirement.
ref-cycles	Total cycles (not affected by CPU frequency scaling).
L1-dcache-(loads/stores/prefetches)	Level 1 Data cache read/write/prefetch accesses.
L1-dcache-(loads/stores/prefetches)-misses	Level 1 Data cache read/write/prefetch misses.
LLC-(loads/stores/prefetches)	Last Level Cache read/write/prefetch accesses.
LLC-(loads/stores/prefetches)-misses	Last Level Cache read/write/prefetch misses.
iTLB-loads	Instruction TLB read accesses.
iTLB-load-misses	Instruction TLB read misses.
node-(loads/stores)	Local memory read/write accesses.
node-loads-misses	Local memory read misses.

5.3.5 Virtual Machine Performance

Virtualized environments present a challenging problem for performance monitoring. Even-though some PMC can be accessed from the host (physical) machine for some virtualization software, such as VMware and KVM; the guest virtual machines (VMs) do not have direct access to the underlying hardware. This may prevent the use of some previously discussed profiling techniques. For instance, Xen does not virtualize the PMC registers due to the significant performance cost that would be incurred, preventing the use of tools like *perf* and *Oprofile* in the guest machine.

Tools such as *Xenoprof*[10] instrument the hypervisor and the guest OS to enable the monitoring of PMCs within the hypervisor, which are then matched to the guests' operations. Since *Xenoprof* is an extension of *Oprofile*, it is capable of providing similar performance analysis functionality. Recent versions of KVM and Qemu also support PMC in the guest OS.

5.4 Interaction between Power and Performance

While the performance characteristics of key system components are generally well understood, the corresponding power characteristics remain less so. This can largely be attributed to differences in component interactions, where performance is often considered independent, power has many flow-on effects, creating interdependencies between compo-

[10]http://xenoprof.sourceforge.net/

Table 5.2 Thermal Design Power (TDP) for servers and desktops' processors provided by vendors data sheets.

Processor	Cores	Max. Freq. (MHz)	TDP (W)
Intel Atom E3815	1	1460	5
AMD Opteron X1150	4	2000	9
Intel Atom E3845	4	1910	10
Intel Core i7-4600U	2	2100	15
AMD Athlon II X2 260u	2	2000	25
Intel Core i7-3615QE	4	2300	45
Intel Core i7-4960HQ	4	2600	47
Intel Xeon E7-4807	6	1860	95
AMD Opteron 8380	4	2500	115
Intel Xeon E7-4860	10	2266	130
Intel Core i7-4960X Extreme Ed.	6	4000	130
AMD Opteron 6386 SE	16	2800	140
AMD FX-9590 Black Edition	8	4700	220

nents. It is only with careful consideration of these effects that the relationship between power and performance can be understood for each individual component. Therefore, this section presents an overview of some of these component-based relationships, along with the corresponding execution states relevant to power modeling.

5.4.1 CPU

The processor is one of the components that consumes the largest amount of power within a system, having a sizable idle value and a broad range of dynamic operating power values. The industry standard for processors' power dissipation comparison is its Thermal Design Power (TDP). This metric defines the maximum power a cooling system is expected to dissipate and may significantly vary between processors' models, going from 5 to 220W on recent models (see table 5.2). For example, the AMD Opteron 8380 is documented as having an average CPU power of 75W, measured as the expected power use while executing a standard set of benchmarks, and a TDP of 115W. However, power is capable of going even higher than the TDP, except a hardware shutoff will be triggered to prevent hardware damage. This illustrates the significant, dynamic range of operating power use that can be experienced by a processor.

Much of the runtime variation is attributed to the diverse utilization levels experienced by running applications of different workload types. All applications lie somewhere on a spectrum between memory-bound and CPU-bound operation. A CPU-bound application performs a large number of calculations during execution, requiring a very small working set of data. This results in high processor utilization, where execution time is dominated by processor operations. Alternatively, a memory-bound application is almost the complete opposite, where execution time is dominated by the time delay of fetching data from main memory or disk. Memory fetches stall processor execution, which remains idle while the

data is retrieved. This results in few calculations been performed, and a correspondingly low processor utilization.

This variation in workload utilization characteristics has been leveraged by the most prevalent power saving policies in modern operating systems. Power is saved by adjusting the processors' operating frequency at runtime using DVFS, so that it is ideally set proportionally to the memory-boundedness of an application. Using a low operating frequency will slow the rate of processor operations, increasing execution time for compute tasks while saving power. Since only processor operations are impacted, a memory-bound workload will experience minimal performance degradation as the memory access latency is not affected. However, running a CPU-bound workload at a low frequency will sacrifice a great deal of performance. Many alternate policies have been implemented to evaluate the tradeoff between power and performance, attempting a maximizing savings without impacting performance. Figure 5.1 shows the power dissipated by a machine equipped with an Intel Core i7-3615QE processor while running Linux's CPU stress benchmark at all available frequencies.

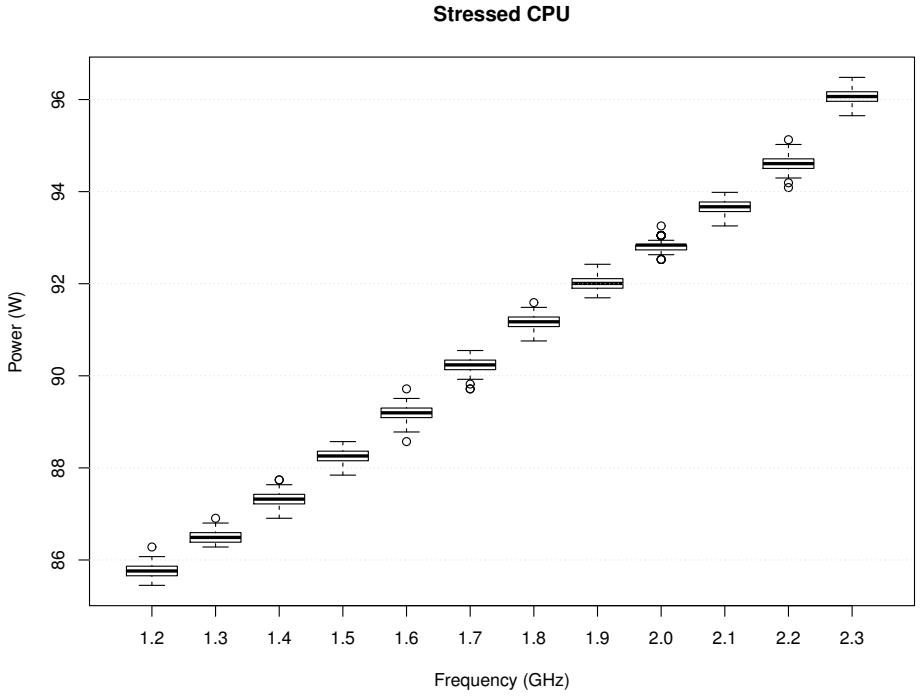

Figure 5.1 Power dissipated by a Intel(R) Core(TM) i7-3615QE CPU machine while running the same workload on different frequencies.

Unfortunately, such policies do not allow for any power savings while executing CPU-bound applications, resulting in high power consumption. To achieve such savings, manufacturers have implemented fine-grained, low-level power saving techniques in hardware, namely Cstates. Aggressive clock gating is used to switch off parts of the microproces-

sor's circuity, temporarily suspending functional units, where state is maintained by stopping flip-flops. Reasonable power savings may be made with the ever increasing density of processors and number of functional units. However, this essentially creates a hidden power state, strictly controlled in hardware, that the operating system is not able to observe. This raises new challenges for modeling power, although if enough fine-grained details are known about the current application workload, the processor's functional unit state could potentially be inferred by the expected unit utilization.

While the processor consumes a large amount of power, it is worth remembering that some of the observed consumption may be due to flow-on effects from other system components. The most straightforward illustration of this is the power use of system fans. As the processor's utilization increases, the temperatures generated increase, causing the cooling system to use more power. However, this relationship is complicated by the thermal changes occurring at a slower rate than the corresponding changes in utilization, as was observed in [38]. Such cross-component power dependencies are more significant for the processor than any other system component, given the central role played in all system operations.

5.4.2 Memory

The interdependence of processor and memory power consumption was indirectly shown in the previous section, where memory-bound workloads stall execution, lowering utilization and power consumption of the processor. However, the power reduction in the processor cannot be easily isolated from any potential increase in power use from the memory modules due the changes occurring simultaneously.

This can lead to the actual power use of memory modules been obscured, causing erroneous conclusions to be drawn as to the significance of power use. For instance, memory is sometimes thought to consume a sizable amount of power, but the specifications for Kingston hyperx memory shows a power draw of only 0.795W[11] while operating. A typical system may contain about eight memory modules, giving a total of 6.36W, which is almost insignificant in the context of other system components.

Observed changes in power consumption for a memory dominated workload can most likely be attributed to cross-component power dependencies, where the memory modules themselves have minimal impact. Instead power reduction comes from lower processor utilization, given that a new task is not switched onto the processor, which will have the further flow-on effect of reducing temperatures.

5.4.3 I/O

A certain percentage of memory accesses will result in pages being fetched from the hard disk, incurring additional processor latency as data transfers are significantly slower than the processor's operating speed. However, any large data transfers will be handled by a direct memory access (DMA) operation, allowing the processor to continue execution while transfers proceed independently.

As a result, the processor will not be aware of the hard disk's operating state at any given time. Therefore, hard disk utilization levels can only be monitored from the operating system, which is able to track the interrupts used to handle the transfer, in addition to bus utilization levels.

[11] http://www.valueram.com/datasheets/khx1600c8d3t1k2_4gx.pdf

Despite the availability of utilization values, workload-specific power is not always modeled if only a small number of hard disks are used in a given system. This is due to the relatively narrow range of power use while running. For instance, the WD VelociRaptor[12], shown in Table 5.3, has a maximum read/write power of 5.8W and an idle of 4.2W, providing a maximum variation of 1.6W. This is a sizable power saving as a percentage of hard disk power, but in the broader context of a server using upwards of 400W, is an insignificant value. A constant value, providing a small estimation error, could be considered an acceptable tradeoff in certain circumstances. A similar case can also be made for the WD SiliconDrive[13] solid state disk, also shown in Table 5.3. However, as the number of hard disks used increases, this error becomes much less acceptable, requiring modeling.

Table 5.3 Power states for two Western Digital hard drives

	WD VelociRaptor Workstation Hard Drive	WD SiliconDrive
Random Read/Write	5.1 Watts	1.0 Watts
Sequential Read/Write	5.8 Watts	1.0 Watts
Idle	4.2 Watts	0.4 Watts
Standby and Sleep	1.1 Watts	

5.4.4 Network

A further source of system power consumption is the network interface card (NIC), which acts like an I/O device in data centers using network storage, requiring data to periodically be transferred from remote servers. Much like the hard disk, the NIC can uses nearly as much power in idle as when active. This is largely due to the requirement for network state to maintained, ensuring availability and responsiveness. For example, the two 10 Gbps ethernet NICs used in [32] consume 21.2 and 18.0 Watts while idle. When active, the power for each card increases by only 0.2W. Surprisingly, the network activity does not seem to affect power consumption as much as might be expected.

Similar results were also observed in the fiber protocol network card that had an offload processor, where no other NIC did. The offload processor caused the idle power to be just over double that of a similar network card which did not have an offload processor. Switching from idle to active showed negligible power difference, indicating the offload processor is not likely to be using any power saving features, such as sleep states.

For the given network card details, the power draw can be modeled as a constant value, as the estimation error will be very small. The maximum estimation improvement will be less than 1%, meaning it is not worth the performance overhead that would potentially be incurred by monitoring network utilization.

[12]http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701284.pdf
[13]http://www.wdc.com/global/products/specs/?driveID=1120&language=1

5.4.5 Idle states

Power models seek to model the power response in components for various utilization levels. However, one utilization level which is often not explicitly modeled [31], is the base power while the system is idle. This is important because in many large-scale server deployments, a significant proportion of time is spent idle [2]. If a model does not incorporate these workload phases, then the model will begin to diverge from real world power use over long periods of operation.

Power models essentially estimate the dynamic response in power for a corresponding change in utilization, specifically the dynamic power of a system. Therefore, the system power, which does not change with utilization is considered static power, representing the base power for the system and components. The static power includes many unmodeled system components, such as the power supply unit or motherboard chipset. At a finer granularity, it can also be thought of including the base power for many of the 'dynamic' components. For instance, the processor consists of both static and dynamic power, where dynamic power is the utilization response and static power is the persistent base power for the component. Figure 5.2 presents the power savings due to the use of these techiniques on a Intel Core i7-3615QE. Considering that its TDP is 45W (table 5.2), these savings can achieve up to 33% of processors' TDP. Furthermore, it will be shown in the model evaluation section, that not making this distinction between what is and is not modeled has the potential to misrepresent the accuracy of estimations.

In summary, different workload types will keep different components busy in the system and therefore incur different power usage patterns. This observation is crucial for building power estimation models based on system events.

5.5 Power Modeling Procedure

The process of deriving a power model, capable of providing accurate runtime power estimates, without the need for special purpose hardware metering, can be broken down to four steps: (1) variable selection, (2) training data acquisition, (3) model construction, and (4) model evaluation. The process followed during each stage of development will ultimately determine the generality and accuracy of the resulting model. In this section, each step is independently discussed, providing general guidance, that can be used to create implementations in a variety of system environments.

5.5.1 Variable Selection

The first step in deriving a power estimation model is the selection of performance events that strongly correlate with power, namely Key Performance Indicators (KPI), for a diverse set of workloads. The available performance events are largely dictated by the chosen modeling methodology and systems execution environment.

For instance, many modeling methodologies restrict performance monitoring to the processor and memory modules, given their dominance of both execution performance and power consumption. In such cases, fine-grained performance events are desired, resulting in the selection of PMCs for their ability to monitor low-level events. This in turn imposes a limit on the number of events able to be monitored simultaneously, due to the microarchitecture dependence of the chosen approach. Alternatively, including network and I/O components in the modeling methodology requires the use of kernel performance events, possibly in addition to PMCs. Therefore, the initial specification of modeling method-

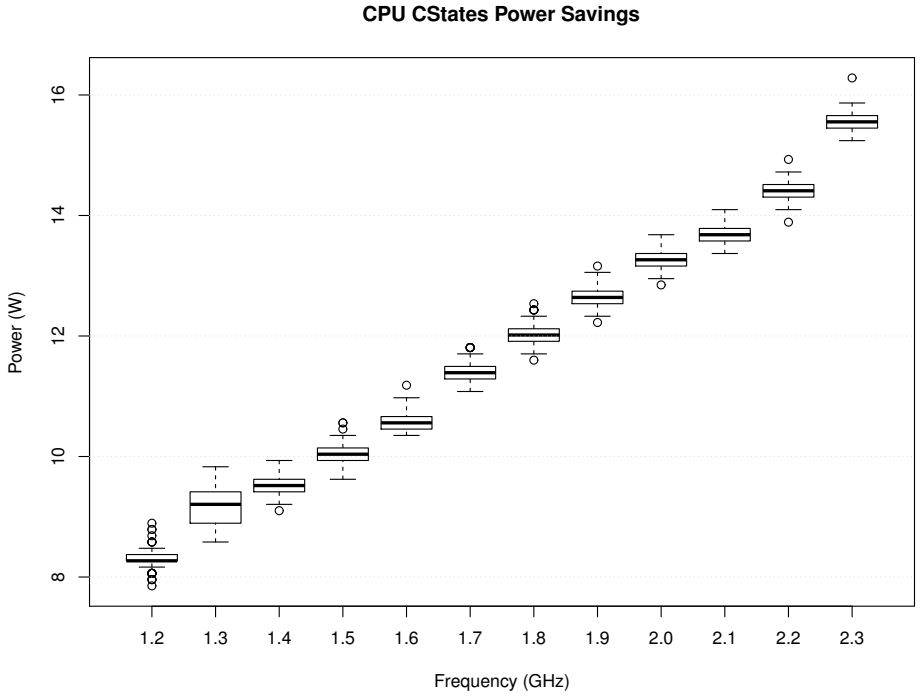

Figure 5.2 Power savings by a Intel(R) Core(TM) i7-3615QE CPU due to processors' idle states (Cstates) on different frequencies.

ology will determine the type of performance event best suited to the desired modeling objectives.

Once the performance monitoring method is selected, an initial dataset is required for determining the strength of correlation for each performance event. The intention of this dataset is to collect a representative sample of performance and power values for each performance event, across a variety of workload types, ensuring a generalized estimation model. The most commonly adopted approach is to periodically collect performance and power values during the execution of an entire benchmark suite, such as SPEC or NPB. By executing the entire suite, the resulting dataset should incorporate different execution characteristics. A more comprehensive approach is sometimes adopted, where a suite of micro-benchmarks are used in place of a benchmark suite. Bertran et al. [7] used a suite of 97 micro-benchmarks to stress every possible power component in turn. Regardless of the approach taken, this training phase can be time consuming for a large number of performance events as the ability to simultaneously monitor counters is limited, requiring multiple iterations of each benchmark to collect all performance events.

Finally, the dataset is used to determine the correlation of each event to power, allowing the top events to be selected. For a large set of performance events, principle component analysis is the best statistical method for finding the principal events. However, a common compromise is to use domain specific knowledge to restrict the scope of performance

events to key units of interest. For example, Singh et al. [31] decomposed the processor and memory into four categories, considered the most significant functional units: FPU, memory accesses, processor stalls and instructions retired. Consequently, only 13 PMCs needed to be collected for the dataset, with a single, strongly correlated event being selected to represent each category. Such a small dataset facilitated simpler, manual correlation methods.

5.5.2 Training Data Collection

While the benchmarks used for event selection provide a representative sample of workload types and utilization levels, they are not guaranteed to represent an exhaustive set of possible values. For a power model to be widely deployable, it needs be capable of accurately estimating the power of previously unseen workloads, requiring benchmark independence. Therefore, micro-benchmarks are exclusively used for collecting model training data.

A specific micro-benchmark is created for each of the selected performance events, with each being configured to explore a diverse spread of performance values. If micro-benchmarks were used for event selection, those corresponding to the selected events can be reconfigured and reused for collecting the model training data. By replicating performance events beyond typical ranges, the resulting model will be more capable of accurately estimating power for less commonly seen applications. However, this requires a tradeoff to be made between accuracy and training time, as a significant number of execution configurations exist. A commonly used compromise is to explore a select sample of boundary cases, while spending the majority of training time within typical ranges.

5.5.3 Learning from Data

The most commonly used approach for modeling power is to derive a linear regression model from a training set of key performance indicators (KPI)/power observations. Training data is periodically collected as a series of tuples, $\langle power, KPI_1, \ldots, KPI_n \rangle$, where the resulting accuracy of the model will depend on how well the training data represents the expected workload type for power estimation. For the regression model, power is the dependent variable estimated by the explanatory performance values.

Let $\mathbf{y}$ be a vector of n power measurements (observations) from the training dataset

$$\mathbf{y} = [y_1, y_2, \ldots, y_n], \tag{5.1}$$

where n is the total number of tuples in the dataset. The performance events are the independent variables used for predicting power, defining a $X_{n \times m}$ matrix, where each matrix line is described as

$$\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,m}] \qquad i \in \{1, \ldots, n\}, \tag{5.2}$$

where $\mathbf{x}_i$ is a vector of m key performance event measurements taken at time t_i. This forms the linear regression model:

$$y_i = \beta_0 + \sum_{j=1}^{m} \beta_j \phi_j x_{i,j} + \xi_i, \tag{5.3}$$

where ξ_i is the measurement error and $\phi_1, \ldots, \phi_m$ are non-linear functions. The β coefficient parameters can be estimated by calculating the least squares

$$S = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{m} \beta_j \phi_j x_{i,j} \right)^2 , \tag{5.4}$$

which minimizes the sum of the squared errors for all of the coefficients.

The resulting regression coefficients, $\beta_0, \ldots, \beta_m$, are used to form the power estimation model, where the runtime measurements of performance event are used as input parameters to the model to estimate the dependent power value. This determines the analytical relationship between a set of key performance events and power. The accuracy of the resulting model will depend on many factors, including the selection of performance events, which will be discussed in more detail throughout this chapter.

5.5.3.1 Alternative Modeling Methods A variety of alternate approaches to power modeling have previously been proposed, varying in the selection of performance events and the correlation methods. Many of the early models were formed by the observed relationship between system utilization, typically measured as instructions per cycle (IPC), and power [37]. Essentially, only the processor power is modeled, but a reasonable estimation accuracy was able to be reached. This was due in a large part to the dominance of processor power in the simple, single-core system.

Mantis [16] extended the relationship between utilization and power to other system components, deriving a power model for the entire system by incorporating memory, hard disk and network performance in addition to processor utilization. The selected performance events come from hardware Performance Monitoring Counters (PMCs) and OS performance events: CPU utilization, off-chip memory access counters, hard disk I/O rate and network I/O rate. The power model is derived from an initial, one-time calibration phase in which benchmarks are run to stress each of these system components in turn, while recording the power from each AC line. A linear program is then used to derive the power model by minimizing the estimation error across all performance events with a linear function. However, the model's accuracy relies on the calibration phase providing a range of utilization levels that are representative of realistic workloads, and not merely the scientific workloads used for model evaluation.

More refined power models are possible by modeling sub-components instead of aggregate component utilization levels. Bellosa [5] noted a strong linear relationship between four synthetic workloads designed to stress individual processor functional units, four logically correlated low-level performance events and power. The four performance events were: processor micro-uops, FLOPS, l2-cache misses and memory transactions. By isolating the power draw of each low-level function, the linear models were able to be combined into a single power model, Joule Watcher.

A similarly fine-grained, processor based, power model was proposed by Singh et al. in [31], which used domain specific knowledge to decompose the processor into four key functional units. A single, strongly correlated PMC was found for each functional unit. Event-specific micro-benchmarks were designed to stress the breadth of potential operating values, taking care to include some extreme outliers. Through this exploration of a large utilization space, it was observed that some performance events experienced different power characteristics at low levels than at high levels, which can only be modeled with a non-linear function. The model was further improved through the use of a piecewise linear function, separating the low event values from high, allowing subtly more complex event

characteristics to be modeled. However, the improvements provided were not explicitly evaluated, merely the accuracy of the resulting model.

The increasing complexity of processor hardware has raised some concerns that linear models may not be sufficiently complex to accurately model power anymore. In fact, these concerns are not new but merely an extension of the ongoing debate regarding the required trade off between model complexity and accuracy. A marginally more complex model was proposed by Bircher et al. [8] where the key subsystems are modeled independently using a quadratic function. The five key subsystems modeled are: CPU, memory, chipset, I/O and disk. The power for each subsystem is determined during an initial training period by placing resistors in series with the power supply, making it a significantly more complicated procedure than the other approaches discussed so far. After the power for each component is isolated, a series of benchmarks are run, stressing each subsystems in turn. The resulting model uses processor-centric performance events, all PMCs with the exception of interrupt counters, to model all of the components. This is a rather unique approach, as many of the previous systems rely on operating system events to monitor utilization levels for components such as the hard disk. Instead, the processor PMCs are used to analyze the "trickle-down" effect of processor events through other system components. For instance, a last-level cache miss will cause a memory access, which may in turn cause a disk access. Monitoring these effects allowed the subsystem components to be modeled using only six performance events, with an average error of less than 9% per subsystem.

A potential problem with using general utilization metrics, like IPC, is that they provide insufficient detail to adequately distinguish between various workload types. This was observed by Dhiman et al. [13] when executing a series of benchmarks in a range of alternate configurations, which resulted in different levels of power use for similar utilization levels. Therefore, the authors propose a modeling technique based upon a rudimentary clustering of workload types using a Gaussian Mixture Model (GMM), with four input performance counters: instructions per cycle (IPC), memory accesses per cycle (MPC), cache transactions per cycle (CTPC) and CPU utilization. The resulting power model was able to outperform two simple, alternative regression-based models during evaluation.

These are but a select few of the wide variety of power modeling methodologies to have previously been proposed. Alternate methodologies differ in the approach taken for performance event selection, collection of training data, modeled system components and the regression method used to derive the power model. Consequently, the remainder of this chapter attempts to remain implementation independent by discussing all relevant concepts at a high-level whenever possible.

5.5.4 Event Correlation

The power model is constructed by quantifying the relationship between the selected performance events and power for an extensive model training dataset. In the model, performance events provide the explanatory variables used to predict the dependent power variable. This arrangement lends itself well to the use of regression techniques to form an estimation model. One of the simplest, and most readily applied regression techniques is to derive a series of per-component regression models, combined into a single model:

$$power = X_{CPU} + X_{memory} + X_{I/O} + \xi \tag{5.5}$$

While such a model is easily understood, given the prevalence of component-specific performance events, the previously discussed impact of cross-component effects is largely neglected. The use of an error term may mitigate some of the impact, but does not resolve

the underlying problem, with the model's simplicity sacrificing accuracy. Support vector regression provides a more robust regression method by fitting a single, non-linear function to a small training dataset. This approach was used in [35].

The accuracy of simple regression models is limited by the use of a single fitting function, required to adequately represent the execution characteristics of widely varying utilization levels and workload types. The resulting estimation model may become overly general, where a more accurate fit could be achieved by characterizing utilization. The variation in high and low utilization behavior was noted by Singh et al. [31]. To rectify these differences, a piecewise regression model was proposed, enabling different estimation functions to be used at each of the utilization levels.

The improvements provided by workload characterization can be further extended by using a workload classification scheme. By classifying execution workloads, a separate regression model can be derived for each source of a dominant workload. For instance, most derived models are processor centric, achieving good estimation accuracy for processor dominated workloads. However, the accuracy can be significantly worse for other workloads, such as memory-bound applications, due to the model's bias towards processor power. Enabling the model to tailor power estimates to a given workload helps to remove much of this bias by using the most appropriate estimation function at the appropriate time. Such an approach was taken in Dhiman et al. [13], where it was observed that similar utilization levels often resulted in different power consumption for various workload types. Classification allowed each workload to be treated separately, improving the overall estimation accuracy.

5.5.5 Model Evaluation Concepts

Power models are evaluated by assessing the effectiveness of power estimates for a suite of benchmarks. For a fair evaluation, the chosen benchmark suite should not be the same as was used during event selection, but should similarly consist of a diverse set of workload types. This helps determine how well a given model will generalize across various utilization levels, as a poorly selected set of performance events and training dataset may contain bias towards a specific workload.

5.5.5.1 Statistical Metrics Summary statistics like the median error are commonly used to present the evaluation error of the estimation model, but this has the potential to misrepresent the accuracy of the model. This is primarily due to the selection of evaluation benchmarks, which are typically scientific workloads, and their dominance of a single execution phase. For example, a compute intensive benchmark will spend much of its execution time in high utilization phases, with a significantly smaller portion of execution time spent executing memory-bound tasks, such as reading/writing log files. As a result, a power estimation model will achieve a low median error overall if it is able to closely estimate the dominant execution phase—compute intensive work—given its dominance. Therefore the median error is not affected by the other execution phases, making it a poor metric to gauge the quality of fit for all execution phases.

Alternatively, the mean absolute error (MAE) provides the mean estimation error for all execution phases, and will thus be negatively impacted by any significant outliers caused by poor estimation during non-dominant phases. However, this only solves part of the problem, as a single metric, such as the mean absolute or median error, is not capable of providing a meaningful description of total execution accuracy. They can only provide cursory insights. More detail can easily be provided by additionally including the standard

deviation, which describes the spread of estimation values from the mean over the entire execution.

5.5.5.2 Static versus Dynamic power The importance of the distinction between static and dynamic power was previously discussed, which is an important consideration to be made when evaluating power estimation. The most important takeaway is that a power model is essentially only estimating the dynamic power of a system. That is the response in power consumption to a corresponding change in performance events or utilization. Presenting the estimation error as a percentage of the system's dynamic power—that is actually being estimated—allows for fairer comparisons to be made between different models, developed on different systems. An example of this was given in [38]:

> For example, suppose there are two different power models with identical mean estimation errors of 5%, but for different machines with an equal total power consumption of 200W. Without further information, the initial conclusion that can be drawn by the reader is both power models are equally accurate. However, a different conclusion can be drawn if it was additionally known that machine 'A' has a dynamic power (aka workload dependent power) of 30%, while machine 'B' has a larger dynamic power of 60%. Given a total error of 10W (5% of total 200W) and the constant static power (aka base power), it is now clear that the model based on machine 'A' has a 16.6% error for dynamic power while the model based on machine 'B' has a much lower error of 8.3% for dynamic power. The extra information on the proportion of static power and dynamic power enables a fair comparison to be made between the two, otherwise apparently identical power estimation models.

Results can appear misleading or lead to erroneous conclusions if they are not presented in the fairest way possible. Furthermore, the actual accuracy of an estimation model can potentially be obscured by the relative static and dynamic power values on systems with a relatively small dynamic power range as a proportion of total power. Such cases are more likely on systems that use low-power components, which can have minimal operating ranges.

5.5.6 Model Evaluation

While it is desirable to achieve perfect power estimation, such an objective may be unobtainable in practice. Such attempts may incur significant runtime overhead, where a less accurate, but more efficient power model, may prove to be sufficient for most use-cases.

5.5.6.1 Estimation errors In modeling hardware component power consumption, a tradeoff between model complexity and accuracy is required. This problem is exacerbated by the increasing complexity of newer microarchitectures, which can be attributed to the incorporation of: hardware threading, fine-grained power saving modes, and increasing component densities. Therefore, estimation models must approximate many of these power factors, balancing runtime performance monitoring and model complexity with estimation accuracy. Collecting a large number of performance events, and modeling all of the corresponding interactions reaches a point where it becomes more akin to simulation, with gains in accuracy being outweighed by the runtime overhead.

Fortunately, if appropriate care is taken while deriving the model, many sources of approximation can be controlled for, or incorporated into the model. For instance, Mair et al. [39] noted that the version of compiler used or the compiler optimization level, impacts a benchmarks power characteristics. Such variations could be incorporated into the model

by treating alternative compiler/benchmark configurations as separate benchmarks used during training, instead of a single benchmark run multiple times. Temperature is another commonly considered factor influencing power consumption, which can be difficult to reliably measure. This is primarily due to the poor quality of standard measurement equipment and significant latencies in observable thermal effects within a system. A simple, but apparently rarely considered, approach to mitigating some of the impact is to ensure that execution times are sufficiently long so that a stable operating temperature is reached before data collection. Short, periodic executions can lead to below-average power use for a given workload, as the temperature will be below the typical operating point given insufficient time for processor warm-up. These two factors, among others, illustrate the importance of taking care in setting up a model's experimental configuration.

However, some sources of estimation error are beyond measurement and are unable to be incorporated into the model. The most common source of these errors is hardware power saving features that operate independently of the operating system, and are therefore transparent to the system. The best example of this, which was previously mentioned, is the anecdotal evidence of aggressive clock gating in modern processors [26], where functional units within the processor are shut off while inactive, in order to save power. Given the lack of understanding as to how this feature operates, it is even difficult to trigger the occurrence of certain actions during model training in an attempt to incorporate the respective power variations. A further source of inconsistent hardware power measurements may be due to manufacturing variability. An experiment in [26] observed an 11% difference in power between two Intel Core i5-540M processors, in the same experimental configuration.This can exacerbate estimation errors where a model is trained on one processor and is used on other processors, thought to be identical. This illustrates the unlikely nature of eliminating all sources of estimation error. Therefore, the question should be, what is an acceptable estimation error?

5.5.6.2 An Acceptable Error There is no single criteria for determining if a given power estimation model is accurate enough, as each use case will have different tolerances of estimation errors. For instance, using the power model to evaluate how best to consolidate workloads in a datacenter has a rather high tolerance, suggested to be 5-10% by [26], due to the coarse-grained nature of per-server power estimation. Alternatively, if the same datacenter were to use this model to charge clients for the power each submitted job uses during execution, an error this high for the entire system would not be tolerated. Such a high error would possibly lead to charging clients for resources not actually used.

The most commonly proposed use for power models is in making power-aware task schedulers. Power-aware task scheduling is a rather generic term, giving rise to many different scheduling policies. For the discussion here, we define "power-aware task scheduling" as using a scheduler that prioritizes resource allocation to the most power efficient uses, while maintaining all strict resource requirements and workload deadlines. For instance, to determine which task should be allocated additional processor cores, the power model can be used to determine the change in power in response to a given allocation. The power efficiency for each task is then evaluated as the size of the relative power change, thereby indicating which task will use the least additional power. A fair evaluation has two accuracy requirements. First, the estimation error needs to remain stable across workloads to prevent any bias in evaluation. Second, the estimation error needs to be less than the power difference of any two cores, otherwise the error may hide the real change in power. These requirements mean the allowable estimation error will be system specific, as different architectures will have processors with significantly varying operating ranges for power

use. Unfortunately, this means that no single rule exists for determining a specific threshold for modeling accuracy. However, the simple fact remains, the greater the estimation accuracy, the better.

5.6 Use Cases

Power estimators can be used on several perspectives. This section presents some of its applications focusing on three possible kinds of users: end-users, software developers and system managers. These usage perspectives provide the motivational background for the creation of power estimators which differ according to the machine's architecture. Here we divide the power modeling methodology based on single core, multicore and distributed systems, summarizing some implementations and their accuracy.

5.6.1 Applications

Depending on the interest/knowledge of the user, the power dissipated by an application/machine may be differently exploited. If we consider the end-user, i.e. a person who interacts with the system without bothering about implementation details, such estimators may provide a user feedback on how much power is dissipated by the system while executing some programs, increasing the awareness of use impact. This can be achieved either *a priori*, using energy efficiency labels, or at execution time, providing an energy per process monitoring solution. While the former faces some disagreement on how the labels should be created, the latter method is already used, mainly by the operating systems of portable devices.

For software developers, hardware's usage deeply depends on how the application is implemented. The impact of an application over its dissipated power includes the libraries that it uses, its coding patterns and the compilers employed. In [29], the authors compare the impact of compiling a similar code using different compilers and programming languages on its overall energy consumption. They used different implementations of the Tower of Hanoi solver and compared their energy spent to solve the same problem size. The codes were written by different authors and their expertise on the programming languages are not taken into account in the evaluation. A similar approach can be done to select which compilation flags and libraries given software may use.

Possibly, due to its highest energy consumption, the system manager is the one who may have the major benefits. The possibility of monitoring applications' power usage may lead to new resource managing policies on different levels. In [36], the authors exploit the energy as a resource on Linux systems, proposing kernel level allocation policies. Some portable OS, such as Google's Android and Apple's iOS, also monitor applications' power use and the battery status can define if software may or may not be launched. At the data center level, the resource allocation and job scheduler can use such information to turn nodes on or off.

5.6.2 Single core systems

Single core systems are the easiest on which to implement the previously explained methods. On such systems, as there is only a single flow of execution, it is simpler to take the necessary measurements. Indeed, there is only one subject as the core is the same as the processor and as the node. It is then possible to correlate measurements from the

network (node level) to measurement from the performance counters. In [12] the authors demonstrate a three step implementation:

1. Monitoring of 165 sensors (system, performance counters);
2. Selection of relevant sensors to model power consumption of a host;
3. Linear regression fitting to obtain the final model.

The second step in this study is present in order to reduce the time needed for the fitting. The detailed algorithm is simple: first find the sensor that has the most correlation with power (see Figure 5.3), then find the one that has the most correlation with the residual, and so on. The study show the model error as a function of the number of variables used. It has to be noted that models for mono-core systems can be quite precise and can reach a precision of a few percentage points using only three well chosen sensors, and can reach a 1% precision using 5 variables. For this study, the workload was created using several synthetic benchmarks: CPU, memory, disk and network.

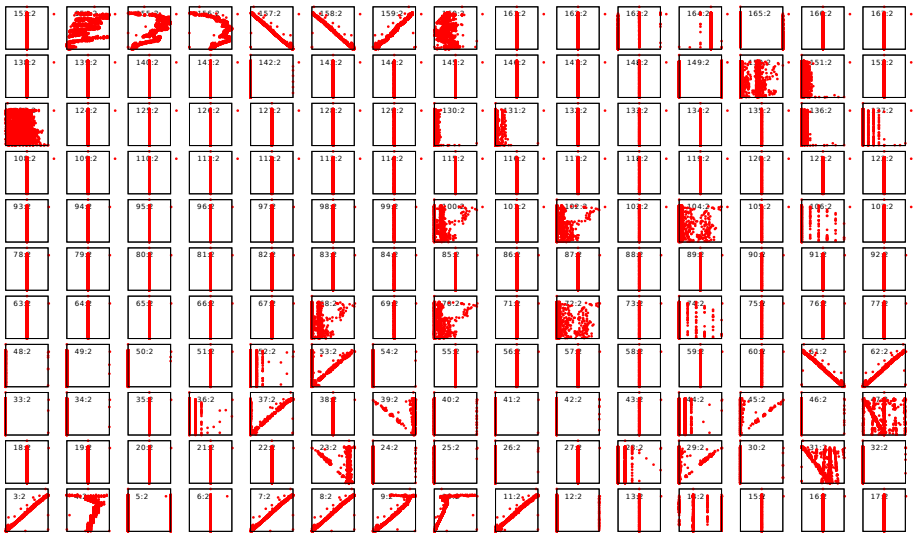

Figure 5.3 Graph of the 165 measured values in function of energy consumption for an synthetic disk benchmark.

5.6.3 Multicore and Multiprocessor

Modeling multicore architectures is not as simple as it may seem. Due to resource sharing, the power dissipated by several cores is not simply the sum of each core. Weaver et al. use RAPL sensors to surpass this issue, evaluating the power of each CPU component with no power model [34]. The authors profile the power of HPC applications by running some benchmarks, e.g. Cholesky decomposition, while monitoring the RAPL register. Although this can be a good approach for HPC environment, where the nodes are fully dedicated

to execute one application at a time, it can only be used on some generations of Intel processors.

Basmadjian and Meer address this issue by modeling each CPU component [4]. These components are stressed at different levels through the execution of micro benchmarks. Hence, the authors propose a power model based on the collected measurements, i.e. observations, including the power saving techniques. The power models are validated using two benchmarks: while-loop and lookbusy. The reported errors are usually less than 5% and always under 9% (3W). The authors also claim that the additive approach (considering the power of a multicore as the sum of several single core CPU) can overestimate the power consumption of the CPU by up to 50%.

5.6.4 Distributed Systems

Recent efforts aim to create/adapt existing data centers to be more energy efficient, by modeling the power consumption. In [3], the authors propose a hierarchical model to estimate the energy consumption of the entire data center, including PSUs, and rack/tower servers at the node level. Even though the authors propose a fine-grained power estimation, there is a lack of validation: the total estimated power of a data center is not compared with its actual consumption. In [9], the authors propose new heuristics to provide an energy-aware service allocation; these heuristics consider not only the energy but also the performance of servers.

CoolEmAll is a European project that addresses the energy efficiency of data centers [6]. Its main goals are to propose new energy-efficient metrics and to provide complete data center monitoring, simulation and visualization software to help aid the design and optimization of new and/or existing data centers. Due to the high cost of data center's cooling system, the heat generation is also considered in job scheduling. CoolEmAll measures hotspots on a datacenter through CFD simulations, providing results that are more accurate than heat matrix simulations. These simulations include a high granularity of information that encloses the estimation of power use of each application, the impact of fans on the airflow, the distribution of nodes in a rack and of rack in a room.

5.7 Available Softwares

Tools for estimating the power consumption of computing devices are widely available as both commercial and open source software systems. These tools vary according to the level (system wide, application or thread) and accuracy of their estimations. All of the software described here is open source, but the accuracy of these software systems is not evaluated.

Powerstat [10] is an Ubuntu package that fetches the power drained by the battery and logs it along with the resource utilization of the machine. The software does not have a power model, and thus can only provide system-wide power consumption information. This is the simplest approach to allow users to have an overall impression of the impact of applications on power consumption, but it requires the use of portable devices and do not work while the devices are plugged into mains power.

PowerTOP [23] is a power estimation tool that monitors not only applications, but also devices such as the display and USB devices (e.g. keyboard, mouse). It identifies the most power consuming applications/devices and proposes some tuning guidelines. PowerTOP uses information gained by interrupting each process in order to estimate its power consumption, i.e. it needs the total power consumption of the machine provided by a power

meter. For portable devices, this power is retrieved from the battery discharging data, while for other devices, it supports using the Extech Power Analyzer/Datalogger (model number 380803). The use of a power meter also allows it to calibrate the power model, providing a more accurate estimation. PowerTOP is written in C++ and requires some kernel configuration options enabled in order function properly, so depending on your system you will need to recompile the kernel in use. It is being ported to Android devices as well.

pTop [15] is a kernel-level tool, that is software-based (i.e. there is no need to run on battery power, or use external power meters). To use pTop, the target system needs to be running at least Linux kernel version 2.6.31, with a kernel patch applied. Energy is composed as the sum of energy used by a set of different components: CPU, memory, disk and the wireless network. For the CPU model, it uses a load-proportional model based on its minimum and maximum power consumption. For the other devices, power-per-event (e.g. memory read, disk access, network upload) data needs to be provided and the device consumption is based on the number of times the event occurred by the energy spent to realize such event. It was originally conceived for laptops. All devices' specifications must be provided by the user (no preexisting calibration data is included).

A similar tool is provided by PowerAPI [20], a Scala-based library that estimates the power at the process level. In contrast to pTop, it contains a CMOS CPU power formula that depends on the CPU's TPD, allowed frequencies and their respective operating voltages. It also contains a graphical interface to plot runtime graphs instead of a top-like interface. As for pTop, PowerAPI requires that the user provides the hardware specifications.

Intel Energy Checker SDK [21] proposes to measure the "greenness" of systems by comparing the dissipated power with the productivity of a system, facilitating energy efficiency analysis and optimizations. It was originally conceived to run in data center or telecom environments, but nowadays the SDK can be used on client or mobile platforms as well. Energy Checker supports a variety of programming languages (e.g. C/C++, C#, Objective-C, Java, PHP, Perl, and other scripting languages) enabling source code instrumentation. It uses different metrics regarding the type of application. These metrics need to have at least two counters: one to provide the amount of useful work, while the other gives the energy consumed while performing this work. This SDK can operate with or without a power meter. If the user does not have a power meter, a CPU proportional power estimator is used. This estimator only considers the variable power drawn by the computing node. Otherwise, it supports a wide variety of power meters.

Ectools [11] aims to aid the research of new power models. It is a set of tools conceived to not only profile and monitor the power dissipated by applications, but also to develop and compare new estimators. Its core library (libec) provides a set of power sensors/estimators. The sensors can either be used as variables for the power models or extended into new sensors. The library provides a CPU proportional power estimator which can be calibrated in the presence of a power meter. Interfaces for accessing ACPI, Plogg and WattsUp Pro power meters data are available, allowing an easy way to access the system-wide dissipated power. Ectools contains three auxiliary tools (1) a data acquisition application (ecdaq) that logs all available system wide sensors while running a given benchmark to allow further data analysis; (2) a processes' top list (ectop) that can be configured to show specific sensors or estimators, enabling the user to compare different power models at runtime and order the list by a specific sensor/estimator; and (3) an application energy profiler (valgreen) that provides the total energy spent during the execution of an application. Ectools is written in C++ and can be used to monitor the power dissipated by VMs.

5.8 Conclusion

By quantifying the observed relationship between key execution performance events and system power consumption, an analytical power model can be derived. Such power models are capable of providing power values at a significantly finer granularity than hardware power metering allows, i.e. providing application or component specific power consumption statistics. The availability of application-specific power values allows for significantly improved runtime power analysis over what has previously been possible.

This chapter presented and discussed the concepts behind, and steps involved in, deriving an analytical power estimation model. These steps included: performance event selection, training data collection, model derivation, and model evaluation. The discussion accompanying each step was purposely kept general in order to avoid details that would make the material too implementation or architecture specific. This in turn enables the high-level concepts presented in the chapter to be broadly applied to power modeling across a diverse set of system configurations and execution environments. However, more specific details were presented during the concluding discussion regarding previously published experimental results, with implementations achieving average estimation errors below 5%. Furthermore, power modeling techniques are increasingly being utilized within widely-available software packages, which is likely to stimulate more general uses of detailed power analysis. In the future, power models are set to become an integral feature in the development of advanced software-based power saving policies.

REFERENCES

1. AMD. *BIOS and Kernel Developer's Guide (BKDG) For AMD Family 10h Processors*, Jan 2013.
2. L.A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.
3. R. Basmadjian, N. Ali, F. Niedermeier, H. de Meer, and G. Giuliani. A methodology to predict the power consumption of servers in data centres. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, e-Energy '11, pages 1–10, New York, NY, USA, 2011. ACM.
4. R. Basmadjian and H. De Meer. Evaluating and modeling power consumption of multi-core processors. In *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, pages 1–10, 2012.
5. F. Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, EW 9, pages 37–42, New York, NY, USA, 2000. ACM.
6. M. Berge, G. Da Costa, A. Kopecki, A. Oleksiak, J.M. Pierson, T. Piontek, E. Volk, and S. Wesner. Modeling and simulation of data center energy-efficiency in coolemall. In Jyrki Huusko, Hermann Meer, Sonja Klingert, and Andrey Somov, editors, *Energy Efficient Data Centers*, volume 7396 of *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin Heidelberg, 2012.
7. R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters.

In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, pages 147–158, New York, NY, USA, 2010. ACM.

8. W.L. Bircher and L.K. John. Complete system power estimation: A trickle-down approach based on performance events. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 158–168, 2007.
9. D. Borgetto, H. Casanova, G. Da Costa, and J.M. Pierson. Energy-aware service allocation. *Future Gener. Comput. Syst.*, 28(5):769–779, May 2012.
10. Canonical Ltd. *Powerstat*, 2012.
11. L.F. Cupertino, G. Costa, A. Sayah, and J.M. Pierson. Energy consumption library. In Jean-Marc Pierson, Georges Da Costa, and Lars Dittmann, editors, *Energy Efficiency in Large Scale Distributed Systems*, Lecture Notes in Computer Science, pages 51–57. Springer Berlin Heidelberg, 2013.
12. G. Da Costa and H. Hlavacs. Methodology of measurement for energy consumption of applications. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 290–297, 2010.
13. G. Dhiman, K. Mihic, and T. Rosing. A system for online power prediction in virtualized environments using gaussian mixture models. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 807–812, 2010.
14. M. Diouri, M.F. Dolz, O. Glück, L. Lefèvre, P. Alonso, S. Catalán, R. Mayo, and E.S. Quintana-Ortí. Solving some mysteries in power monitoring of servers: Take care of your wattmeters! In *Energy Efficiency in Large Scale Distributed Systems*, Lecture Notes in Computer Science, pages 3–18. Springer Berlin Heidelberg, 2013.
15. T. Do, S. Rawshdeh, and W. Shi. pTop: A Process-level Power Profiling Tool. In *HotPower '09: Proceedings of the Workshop on Power Aware Computing and Systems*, New York, NY, USA, October 2009. ACM.
16. D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Workshop on Modeling Benchmarking and Simulation (MOBS)*, 2006.
17. S.L. Graham, P.B. Kessler, and M.K. Mckusick. Gprof: A call graph execution profiler. *SIGPLAN Not.*, 17(6):120–126, June 1982.
18. A. Greenberg, J. Hamilton, D.A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008.
19. C.H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. *SIGPLAN Not.*, 38(5):38–48, May 2003.
20. INRIA. *PowerAPI*, 2013.
21. Intel Corporation. *Intel Energy Checker Software Developer Kit User Guide*, 2010.
22. Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2*, Sept 2013.
23. Intel Corporation. *PowerTop*, 2013.
24. R.P. Larrick and K.W. Cameron. Consumption-based metrics: From autos to IT. *Computer*, 44(7):97–99, 2011.
25. C.K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, June 2005.
26. J.C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A.C. Snoeren, and R.K. Gupta. Evaluating the effectiveness of model-based power characterization. In

Proceedings of the 2011 USENIX conference on USENIX annual technical conference, USENIXATC'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

27. A. Milenkovic, M. Milenkovic, E. Jovanov, D. Hite, and D. Raskovic. An environment for runtime power monitoring of wireless sensor network platforms. In *System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on*, pages 406–410, 2005.
28. A. Narayan and S. Rao. Power-aware cloud metering. *IEEE Transactions on Services Computing*, 99(PrePrints):1, 2013.
29. A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. A Preliminary Study of the Impact of Software Engineering on GreenIT. In *First International Workshop on Green and Sustainable Software*, pages 21–27, Zurich, Suisse, June 2012.
30. E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *Micro, IEEE*, 32(2):20–27, 2012.
31. K. Singh, M. Bhadauria, and S.A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009.
32. R. Sohan, A. Rice, A.W. Moore, and K. Mansley. Characterizing 10 gbps network interface energy consumption. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 268–271, 2010.
33. B. Tomlinson, M.S. Silberman, and J. White. Can more efficient IT be worse for the environment? *Computer*, 44(1):87–89, 2011.
34. V.M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 262–268, 2012.
35. H. Yang, Q. Zhao, Z. Luan, and D. Qian. iMeter: An integrated VM power model based on performance profiling. *Future Generation Computer Systems*, 2013.
36. H. Zeng, C.S. Ellis, A.R. Lebeck, and A. Vahdat. ECOSystem: managing energy as a first class operating system resource. *SIGOPS Oper. Syst. Rev.*, 36(5):123–132, October 2002.
37. T. Li, and L.K. John. Run-time modeling and estimation of operating system power consumption. *In ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, pp. 160-171. ACM, 2003.
38. J. Mair, Z. Huang, D. Eyers, and H. Zhang. Myths in PMC-Based Power Estimation. *In Energy Efficiency in Large Scale Distributed Systems*, pp. 35-50, 2013.
39. J. Mair, D. Eyers, Z. Huang, and H. Zhang. Myths in PMC-based Power Estimation. *Under review*, 2013.